Umělá inteligence I

Roman Barták, KTIML

roman.bartak@mff.cuni.cz
http://ktiml.mff.cuni.cz/~bartak

8

Dnes

- Dnes se podíváme na agenty, kteří si vytvářejí **reprezentaci** světa, **odvozují** z ní novou informaci, kterou potom použijí k rozhodnutí, co dělat dál.
- Řeč budu o **logických (znalostních) agentech**, kteří kombinují obecné znalosti s aktuálním pozorováním světa, aby odhalili skryté aspekty současné stavu a použili je při výběru další akce.
- K tomu potřebujeme znát:
  - **jak reprezentovat znalosti**
  - jak přivést znalosti k životu pomocí **uvažování**

Znalostní agent

- Znalostní agent používá **bázi znalostí** – množina vět v daném jazyce – do které přidává další znalosti operací TELL a získává z ní informace operací ASK.
- Odpovědi na operace ASK jsou **odvozeny** od toho, co bylo do báze znalostí uloženo operacemi TELL.
  - odvození může vytvářet nové věty, které nebyly do báze znalostí explicitně zadány (tím se báze znalostí liší od databáze)

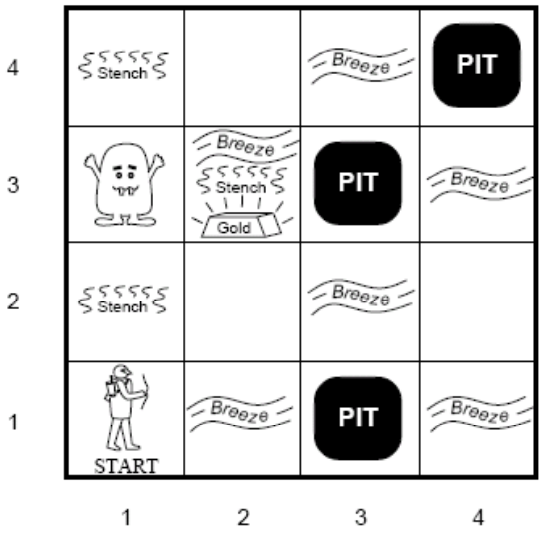
```
function KB-AGENT(percept) returns an action
  static: KB, a knowledge base
          t, a counter, initially 0, indicating time
  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  action ← ASK(KB, MAKE-ACTION-QUERY(t))
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action
```

do báze znalostí ukládáme informace o pozorování světa i o vlastních akcích

odvozování nám umožní vybrat nejvhodnější akci i v případě neúplné informace o světě

Svět Wumpus

- Jeskynní bludiště, kde žije zapáchající **příšera** Wumpus, která každého sežere, jsou tam hluboké **díry**, kde každý uvízne, a jedna hrouda **zlata**, kterou hledáme.



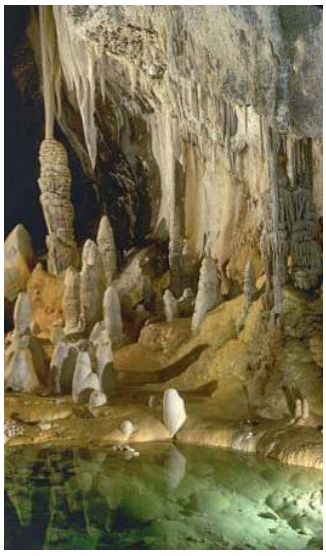
- Wumpus je cítit v nejbližších okolních „místnostech" (ne po diagonále).
- Díry se projevují vánkem v nejbližších okolních místnostech.
- Zlato se třpytí v místnosti, kde se nachází.
- Přes stěny nelze procházet a náraz je cítit.
- Wumpuse lze zastřelit lukem, ale k dispozici je pouze jeden šíp (proletí všechny místnosti ve směru pohledu agenta až ke stěně nebo k zásahu Wumpuse).
  - Zasažený Wumpus vydá skřek, který je slyšet všude, a pak hned zemře.
  - Mrtvý Wumpus stále zapáchá, ale už nikoho nesežere.

Svět Wumpus
z pohledu agenta

- **Výkon**
  - +1000 bodů za sebrané zlato
  - -1000 bodů za spadnutí do jámy nebo sežrání Wumpusem
  - -1 bod za provedení libovolné akce
  - -10 bodů za vystřelený šíp
- **Prostředí**
  - 4 × 4 místnosti, agent začíná na [1,1] pohledem doprava
- **Akční členy**
  - krok dopředu, otočení o 90° vlevo/vpravo, vystřelení, uchopení
- **Senzory**
  - zápach ano/ne, vánek ano/ne, třpyt ano/ne, náraz ano/ne, skřek ano/ne

Svět Wumpus
z pohledu prostředí

- **Plná pozorovatelnost?**
  - NE, vidíme pouze své okolí (částečná pozorovatelnost)
- **Deterministické?**
  - ANO, výsledek akce je jasně daný
- **Episodické?**
  - NE, záleží na pořadí akcí (sekvenční)
- **Statické?**
  - ANO, Wumpus ani díry se nehýbají
- **Diskrétní?**
  - ANO
- **Jeden agent?**
  - ANO, Wumpus se (zatím) nechová jako agent, ale jako vlastnost prostředí



Svět Wumpus

hledáme zlato

nic necítím, nikde nefouká ⇒ jsem v pohodě a můžu jít kam chci

1.

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 OK	2,2	3,2	4,2
1,1 A OK	2,1 OK	3,1	4,1

A = Agent
B = Breeze
G = Glitter, Gold
OK = Safe square
P = Pit
S = Stench
V = Visited
W = Wumpus

tady nějak fouká ⇒ někde tady bude díra, raději se vrátím

2.

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 OK	2,2 P?	3,2	4,2
1,1 V OK	2,1 A B OK	3,1 P?	4,1

tady něco smrdí ⇒ někde bude Wumpus

3.

1,4	2,4	3,4	4,4
1,3 W!	2,3	3,3	4,3
1,2 A S OK	2,2 OK	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P!	4,1

na [1,1] to nebude, tam už jsem byl

na [2,2] to také nebude, to bych ho cítil, když jsem byl na [2,1]

Wumpus je na [1,3]

necítím vítr ⇒ [2,2] bude bezpečná, jdu tam (díra je na [3,1])

...

tady se to pěkně třpytí ⇒ jsem za vodou ☺

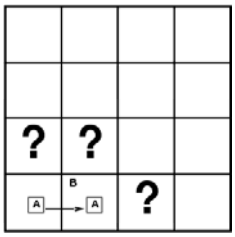
5.

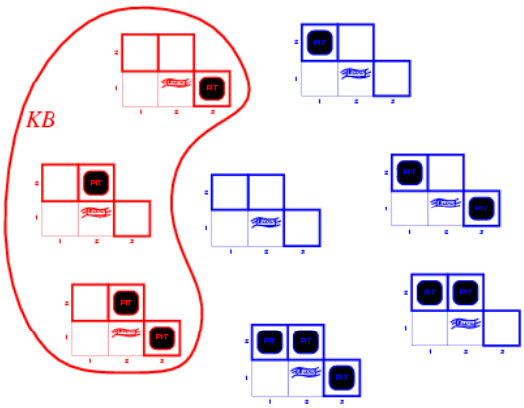
1,4	2,4 P?	3,4	4,4
1,3 W!	2,3 A S G B	3,3 P?	4,3
1,2 S V OK	2,2 V OK	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P!	4,1

Svět Wumpus

modely báze znalostí

- Uvažujme situaci, kdy jsme na políčku [1,1] nic nedetekovali, přešli jsme doprava na [2,1] a tam cítíme vánek.





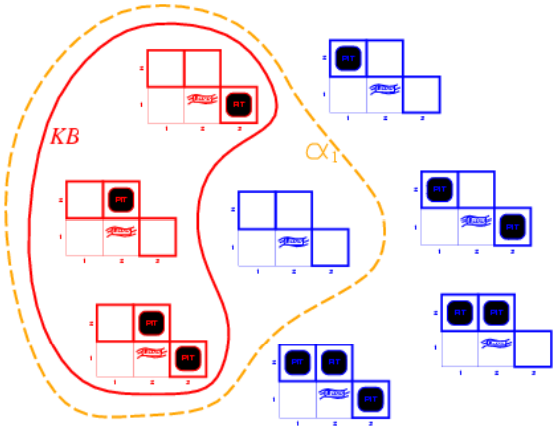
- Pokud uvažujeme pouze detekci děr, máme 8 ($=2^3$) možných modelů (stavů světa).
- Pouze tři modely odpovídají naší bázi znalostí, protože ostatní modely jsou ve sporu s pozorováními:
  - žádná detekce na [1,1]
  - vánek na [2,1]

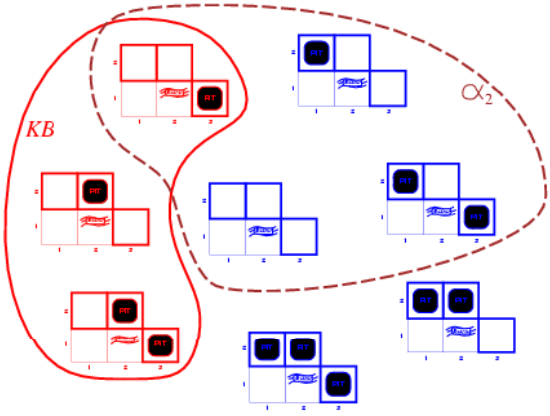
Svět Wumpus

důsledky báze znalostí

- zeptejme se nyní, zda je pozice [1,2] bezpečná
- platí, že α_1 = „[1,2] je bezpečné“ je důsledkem naší báze znalostí?
- porovnáme modely pro KB a α_1
- každý model KB je modelem α_1 tj. α_1 je důsledkem KB

- a co bezpečnost pozice [2,2]?
- porovnáme modely pro KB a α_2
- některé modely pro KB nejsou modely pro α_2
- α_2 tedy není důsledkem KB a nemáte tak jistotu, že [2,2] je bezpečná pozice



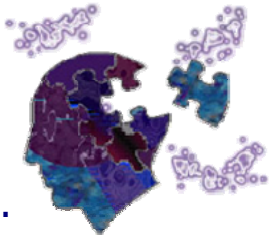


Obecné odvozování

Jak budeme realizovat odvozování z báze znalostí obecně?

Použijeme **výrokovou logiku**,
tj. věty jsou výrokové formule a báze znalostí je konjunkcí těchto formulí.

- **Výrokové proměnné** popisují „mapu světa“
  - $P_{i,j}$ = true právě když je na pozici [i, j] díra
  - $B_{i,j}$ = true právě když na pozici [i, j] cítíme vánek
- **Výrokové formule** popisují
  - známé informace o světě
    - $\neg P_{1,1}$ na pozici [1, 1] není díra (protože tam stojíme)
  - obecné znalosti o vlastnostech světa (např. pokud cítíme vánek, musí být na některé z okolních pozic díra)
    - $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
    - $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$
    - ...
  - získaná pozorování
    - $\neg B_{1,1}$ na pozici [1, 1] nefouká
    - $B_{2,1}$ na pozici [2, 1] fouká
- Použijeme **odvozovací postupy** pro výrokovou logiku.



Výroková logika

v kostce

- **Syntax** definuje povolené věty.
  - výroková proměnná (případně konstanty true a false) je (atomická) věta
  - spojení vět pomocí logických spojek $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$ je (složená) věta
- **Sémantika** definuje pravdivost každé věty vzhledem k libovolnému světu (modelu).
  - **model** je přiřazení hodnot true/false do všech výrokových proměnných
  - P je pravda v každém modelu obsahujícím P=true
  - sémantika složených vět je určena pravdivostní tabulkou

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
$false$	$false$	$true$	$false$	$false$	$true$	$true$
$false$	$true$	$true$	$false$	$true$	$true$	$false$
$true$	$false$	$false$	$false$	$true$	$false$	$false$
$true$	$true$	$false$	$true$	$true$	$true$	$true$

Výroková logika

důsledek a odvození

- M je **modelem věty** α, pokud je α pravda v M.
  - Množinu všech modelů pro α značíme M(α).
- **logický důsledek: KB** $\models \alpha$
  říká, že α logicky plyne z KB
  - nastává právě tehdy, když M(KB) $\subseteq$ M(α)
- Nás budou zajímat **odvozovací metody**, které hledají/ověřují důsledky KB.
  - KB $\vdash_i \alpha$ říká, že algoritmus i odvozuje větu α z KB
  - algoritmus je **korektní** pokud KB $\vdash_i \alpha$ implikuje KB $\models \alpha$
  - algoritmus je **úplný** pokud KB $\models \alpha$ implikuje KB $\vdash_i \alpha$

Odvozovací metody

- Existující dvě základní třídy odvozovacích algoritmů.
  - **ověřování modelů** (model checking)
    - enumerace pravdivostní tabulky
    - Davis-Putnam-Logemann-Loveland (DPLL)
    - lokální prohledávání (minimalizace konfliktů)
  - **odvozovací pravidla** (inference rules)
    - hledání důkazu aplikací odvozovacích pravidel
    - rezoluční metoda

Enumerace

```
function TT-ENTAILS?(KB, α) returns true or false
    symbols ← a list of the proposition symbols in KB and α
    return TT-CHECK-ALL(KB, α, symbols, [])

function TT-CHECK-ALL(KB, α, symbols, model) returns true or false
    if EMPTY?(symbols) then
        if PL-TRUE?(KB, model) then return PL-TRUE?(α, model)
        else return true
    else do
        P ← FIRST(symbols); rest ← REST(symbols)
        return TT-CHECK-ALL(KB, α, rest, EXTEND(P, true, model)) and
               TT-CHECK-ALL(KB, α, rest, EXTEND(P, false, model))
```

Svět Wumpus
α_1 = „[1,2] je bezpečné" = „$\neg P_{1,2}$" je logickým důsledkem KB, protože $P_{1,2}$ je pro modely KB vždy false, tj. [1,2] neobsahuje díru

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	KB	α_1
false	false	false	false	false	false	false	false	true
false	false	false	false	false	false	true	false	true
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
false	true	false	false	false	false	false	false	true
false	true	false	false	false	false	true	**true**	**true**
false	true	false	false	false	true	false	**true**	**true**
false	true	false	false	false	true	true	**true**	**true**
false	true	false	false	true	false	false	false	true
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
true	true	true	true	true	true	true	false	false

- Jednoduše prozkoumáme všechny modely metodou **generuj a testuj**.
- Pro každý model, kde je pravda KB, musí být pravda také α.

Trochu logiky

- Věta (formule) je **splnitelná**, pokud je pravdivá v nějakém modelu.
  Příklad: $A \vee B$, C
- Věta (formule) je **nesplnitelná**, pokud není pravdivá v žádném modelu.
  Příklad: $A \wedge \neg A$
- Logický důsledek lze převést na testování splnitelnosti následujícím způsobem: **$KB \models \alpha$ právě tehdy když $(KB \wedge \neg\alpha)$ je nesplnitelná.**
  - důkaz pomocí zamítnutí
  - důkaz sporem
- Ověřování, zda α je logickým důsledkem KB tedy můžeme převést na problém testování splnitelnosti logické formule $(KB \wedge \neg\alpha)$.
  Typicky se používají formule v **konjunktivním normálním tvaru** (CNF)
  - **literál** je výroková proměnná nebo její negace
  - **klauzule** je disjunkce literálů
  - **formule** v CNF je konjunkce klauzulí

  Příklad: $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$

  Každou větu (formuli) lze převést na CNF.

$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
$(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$
$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$
$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

DPLL

- Davis, Putnam, Logemann, Loveland
  - úplný a korektní algoritmus pro testování splnitelnosti CNF formule (najde její model)

function DPLL-SATISFIABLE?(s) **returns** *true* or *false*
  inputs: s, a sentence in propositional logic
  clauses ← the set of clauses in the CNF representation of s
  symbols ← a list of the proposition symbols in s
  return DPLL(*clauses*, *symbols*, [])

function DPLL(*clauses*, *symbols*, *model*) **returns** *true* or *false*
  if every clause in *clauses* is true in *model* **then return** *true*
  if some clause in *clauses* is false in *model* **then return** *false*
  P, *value* ← FIND-PURE-SYMBOL(*symbols*, *clauses*, *model*)
  if *P* is non-null **then return** DPLL(*clauses*, *symbols*–*P*, [*P* = *value*|*model*])
  P, *value* ← FIND-UNIT-CLAUSE(*clauses*, *model*)
  if *P* is non-null **then return** DPLL(*clauses*, *symbols*–*P*, [*P* = *value*|*model*])
  P ← FIRST(*symbols*); *rest* ← REST(*symbols*)
  return DPLL(*clauses*, *rest*, [*P* = *true*|*model*]) **or**
    DPLL(*clauses*, *rest*, [*P* = *false*|*model*])

Brzká terminace pro částečný model
- klauzule je pravdivá, pokud je libovolný literál pravdivý
- formule je nepravdivá, pokud je libovolná klauzule nepravdivá

Heuristika ryzí proměnné
- proměnná je ryzí, pokud jsou všechny její výskyty se stejným „znaménkem"
- odpovídající literál je nastaven na true

Heuristika jednotkové klauzule
- klauzule je jednotková, pokud obsahuje jediný literál
- tento literál je nastaven na true

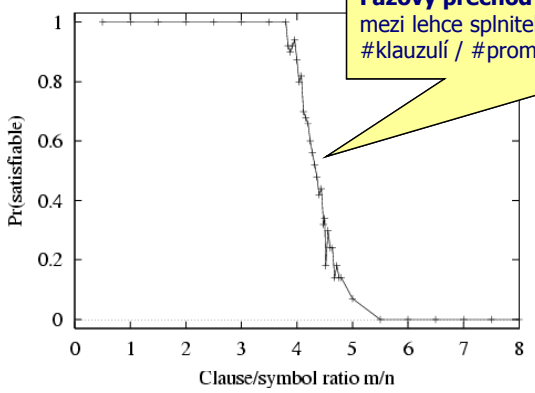
Větvení pro backtracking

WalkSAT

- **Horolezecká metoda kombinovaná s náhodnou procházkou**
  - minimalizujeme počet konfliktních (nesplněných) klauzulí
  - krok je realizován překlopením hodnoty vybrané proměnné
  - **korektní, neúplný algoritmus**

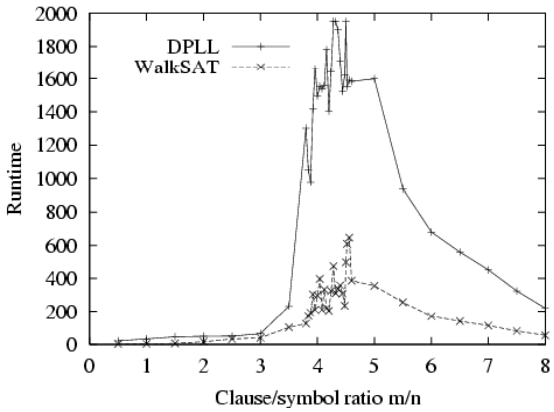
```
function WalkSAT(clauses, p, max-flips) returns a satisfying model or failure
  inputs: clauses, a set of clauses in propositional logic
          p, the probability of choosing to do a "random walk" move
          max-flips, number of flips allowed before giving up

  model ← a random assignment of true/false to the symbols in clauses
  for i = 1 to max-flips do
      if model satisfies clauses then return model
      clause ← a randomly selected clause from clauses that is false in model
      with probability p flip the value in model of a randomly selected symbol
          from clause
      else flip whichever symbol in clause maximizes the number of satisfied clauses
  return failure
```


WalkSAT vs. DPLL



Fázový přechod
mezi lehce splnitelnými a lehce vyvratitelnými problémy
#klauzulí / #proměnných = 4.3

- Náhodné 3-SAT problémy pro 50 proměnných
  - klauzule obsahuje 3 různé proměnné
  - proměnná v klauzuli je v negované podobě s 50% pravděpodobností



- Medián času na vyřešení problému (pro 100 splnitelných problémů)
  - DPLL je docela efektivní
  - WalkSAT je ještě rychlejší

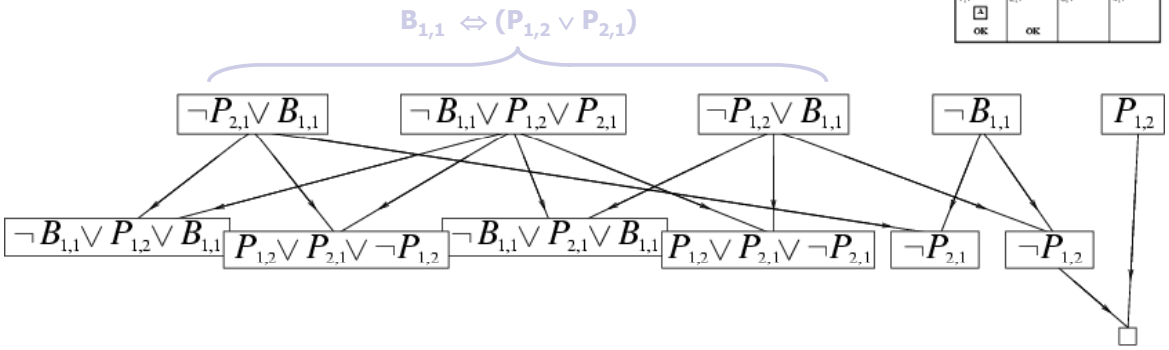
Obecná rezoluce

- Rezoluční algoritmus dokazuje nesplnitelnost formule (KB $\wedge \neg\alpha$) převedené na CNF opakovanou aplikací **rezolučního pravidla**, které ze dvou klauzulí sdílejících komplementární literály (P a $\neg$P) udělá novou klauzuli:

$$\frac{\ell_1 \vee \ldots \vee \ell_k \qquad m_1 \vee \ldots \vee m_n}{\ell_1 \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \vee \ldots \vee \ell_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n}$$

  kde ℓ_i a m_j jsou komplementární literály

- Algoritmus končí pokud
  - nelze přidat žádnou další klauzuli (potom $\neg$ KB $\models \alpha$)
  - vznikla prázdná klauzule (potom KB $\models \alpha$)
- **úplný a korektní algoritmus**



Obecná rezoluce

algoritmus

- Pro každé dvě klauzule vytvoří všechny možné rezolventy, které přidá do množiny klauzulí pro další rezoluci.
  - prázdná klauzule odpovídá false (prázdná disjunkce)
    - ↳ formule je nesplnitelná
  - dosažen pevný bod (nevznikly nové klauzule)
    - ↳ formule je splnitelná, najdeme model
      - postupně bereme výrokové proměnné P_i
      - máme-li klauzuli s $\neg P_i$ takovou, že ostatní literály jsou nepravda při ohodnocení zvoleném pro $P_1,\ldots,P_{i-1}$, potom do P_i přiřaď false
      - jinak do P_i přiřaď true

```
function PL-RESOLUTION(KB, α) returns true or false
    clauses ← the set of clauses in the CNF representation of KB ∧ ¬α
    new ← { }
    loop do
        for each C_i, C_j in clauses do
            resolvents ← PL-RESOLVE(C_i, C_j)
            if resolvents contains the empty clause then return true
            new ← new ∪ resolvents
        if new ⊆ clauses then return false
        clauses ← clauses ∪ new
```

Hornovské klauzule

- Řada bází znalostí obsahuje pouze klauzule, které mají speciální tvar, tzv. **Hornovské klauzule**.
  - obsahují maximálně jeden pozitivní literál

  Příklad: $C \wedge (\neg B \vee A) \wedge (\neg C \vee \neg D \vee B)$
  - typicky vznikají z bází dat, kde jsou všechny věty ve tvaru implikace (například Prolog bez proměnných)

  Příklad: $C \wedge (B \Rightarrow A) \wedge (C \wedge D \Rightarrow B)$
- Budeme řešit problém, zda je daná výroková proměnná – **dotaz** – odvoditelná z báze znalostí skládající se pouze z Hornovských klauzulí.
  - je možné použít speciální variantu rezolučního principu, která běží v lineárním čase vzhledem k velikosti KB
  - **dopředné řetězení** (od faktů ke všem závěrům)
  - **zpětné řetězení** (od dotazu k faktům)

Dopředné řetězení

- Z dostupných faktů odvozujeme pomocí Hornovských klauzulí všechny možné závěry dokud vznikají nové fakty resp. nedokážeme platnost dotazu.
- Je to **metoda řízená daty**.

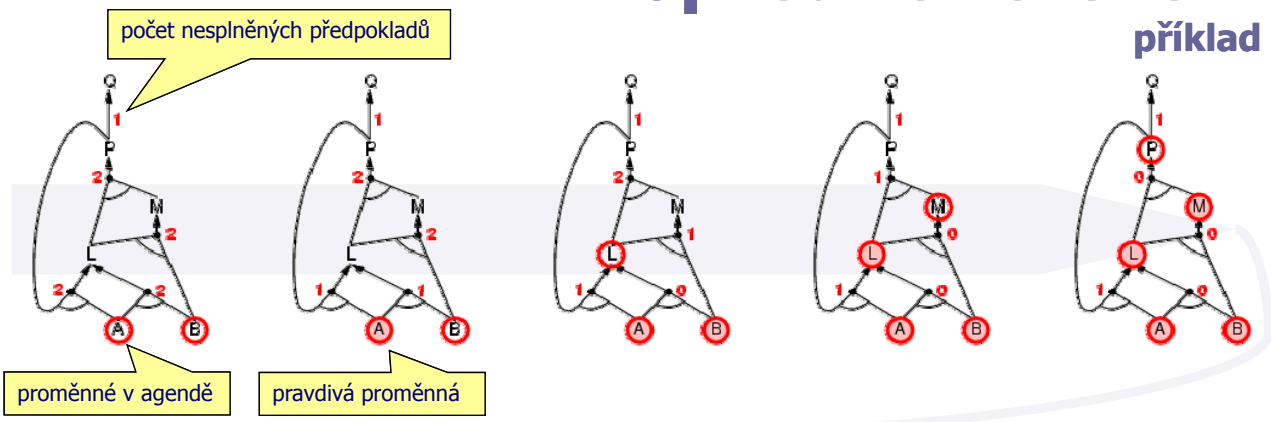
```
function PL-FC-ENTAILS?(KB, q) returns true or false
  local variables: count, a table, indexed by clause, initially the number of premises
                   inferred, a table, indexed by symbol, each entry initially false
                   agenda, a list of symbols, initially the symbols known to be true

  while agenda is not empty do
      p ← POP(agenda)
      unless inferred[p] do
          inferred[p] ← true
          for each Horn clause c in whose premise p appears do
              decrement count[c]
              if count[c] = 0 then do
                  if HEAD[c] = q then return true
                  PUSH(HEAD[c], agenda)
  return false
```

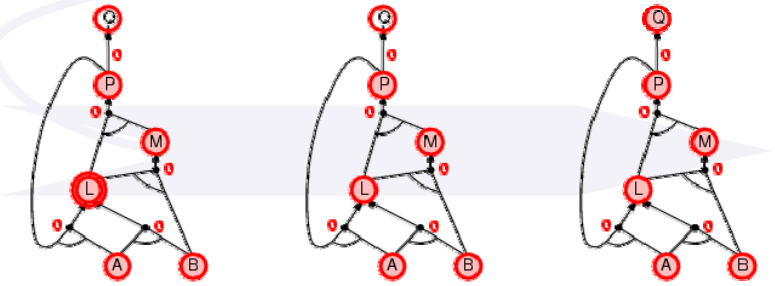
U každé klauzule si pamatujeme počet předpokladů, který zmenšujeme, když je předpoklad dokázán. Klauzule s nula (zbývajícími) předpoklady slouží k dokázání hlavy klauzule

- **korektní a úplný** algoritmus pro Hornovské klauzule
- **lineární časová složitost** v rozměru báze znalostí

Dopředné řetězení

příklad





Báze znalostí a její grafové znázornění

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

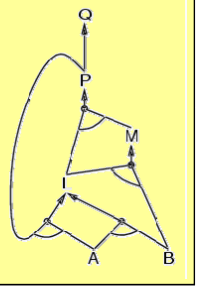
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

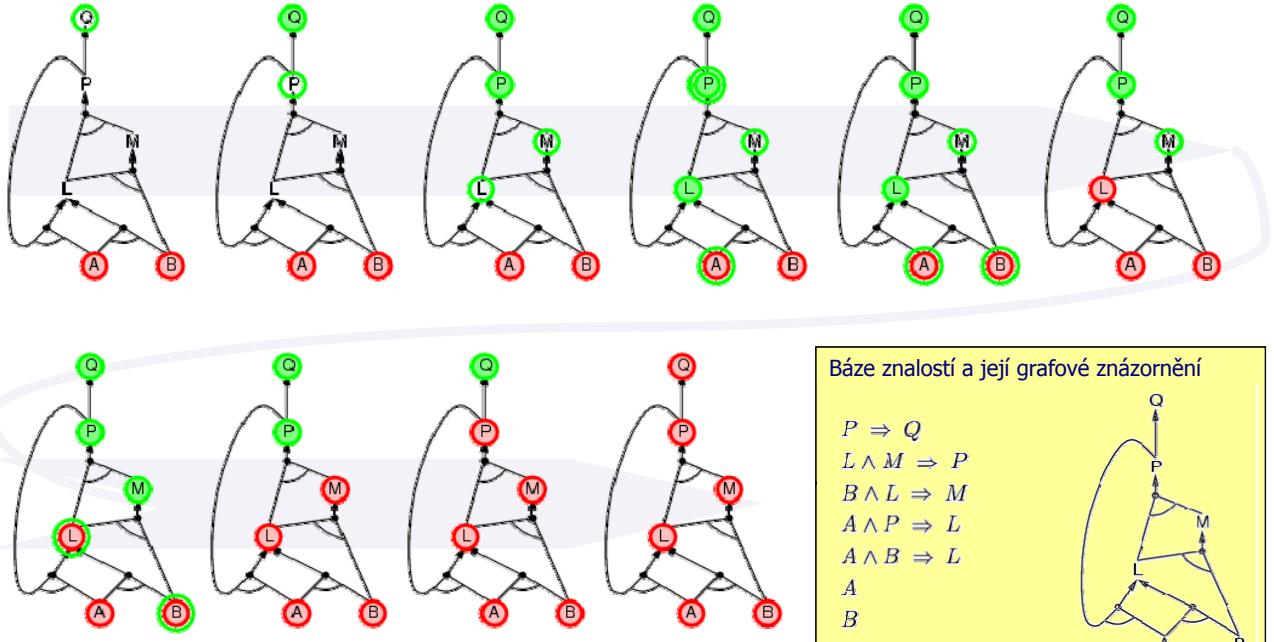
A

B



Zpětné řetězení

- Pro daný dotaz hledáme rozklad pomocí Hornovské klauzule na pod-dotazy, které řešíme stejným způsobem.
- **Metoda řízená dotazem**.



Báze znalostí a její grafové znázornění

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Agent Wumpus
báze znalostí

- Pro jednoduchost budeme v bázi znalostí zatím reprezentovat jen „**fyziku**" **světa** Wumpus.
  - na úvod víme, že
    - $\neg P_{1,1}$
    - $\neg W_{1,1}$
  - také víme, kde a proč vzniká vánek,
    - $B_{x,y} \Leftrightarrow (P_{x,y+1} \vee P_{x,y-1} \vee P_{x+1,y} \vee P_{x-1,y})$
  - jak se v místnostech objevuje zápach
    - $S_{x,y} \Leftrightarrow (W_{x,y+1} \vee W_{x,y-1} \vee W_{x+1,y} \vee W_{x-1,y})$
  - a na závěr tady máme „skrytou" informaci, že Wumpus je právě jeden
    - $W_{1,1} \vee W_{1,2} \vee \ldots \vee W_{4,4}$
    - $\neg W_{1,1} \vee \neg W_{1,2}$
    - $\neg W_{1,1} \vee \neg W_{1,3}$
    - ...
- Správně bychom měli také sledovat **agenta**.
  - kde se agent přesně nachází
    - $L_{1,1}$
    - FacingRight^1
  - a co se stane provedením různých akcí
    - $L^t_{x,y} \wedge \text{FacingRight}^t \wedge \text{Forward}^t \Rightarrow L^{t+1}_{x+1,y}$
    - potřebujeme horní limit na počet časových kroků a i tak dostaneme obrovské množství formulí

Agent Wumpus
algoritmus

function PL-WUMPUS-AGENT(*percept*) **returns** an *action*
  inputs: *percept*, a list, [*stench*, *breeze*, *glitter*]
  static: *KB*, initially containing the "physics" of the wumpus world
    x, *y*, *orientation*, the agent's position (init. [1,1]) and orient. (init. *right*)
    visited, an array indicating which squares have been visited, initially *false*
    action, the agent's most recent action, initially null
    plan, an action sequence, initially empty

  update *x*, *y*, *orientation*, *visited* based on *action*
  if *stench* **then** TELL(*KB*, $S_{x,y}$) **else** TELL(*KB*, $\neg S_{x,y}$)
  if *breeze* **then** TELL(*KB*, $B_{x,y}$) **else** TELL(*KB*, $\neg B_{x,y}$)
  if *glitter* **then** *action* ← *grab*
  else if *plan* is nonempty **then** *action* ← POP(*plan*)
  else if for some fringe square [*i*,*j*], ASK(*KB*, $(\neg P_{i,j} \wedge \neg W_{i,j})$) is *true* **or**
      for some fringe square [*i*,*j*], ASK(*KB*, $(P_{i,j} \vee W_{i,j})$) is *false* **then do**
    plan ← A*-GRAPH-SEARCH(ROUTE-PB([*x*, *y*], *orientation*, [*i*,*j*], *visited*))
    action ← POP(*plan*)
  else *action* ← a randomly chosen move
  return *action*

Informace o výsledku pozorování pro bázi znalostí

Najdeme buňku z okraje, která je bezpečná.

Nebo alespoň buňku, která není prokazatelně nebezpečná.

Najdeme posloupnost akcí, která nás dovede k této buňce přes již navštívené stavy.